applied sciences

MDPI

Article

On the Security of Practical Mail User Agents against Cache Side-Channel Attacks †

Hodong Kim [1], Hyundo Yoon [1], Youngjoo Shin [2] and Junbeom Hur [1,*]

[1] Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea; hdkim@isslab.korea.ac.kr (H.K.); hdyoon@isslab.korea.ac.kr (H.Y.)
[2] School of Computer and Information Engineering, Kwangwoon University, Seoul 01897, Korea; yjshin@kw.ac.kr
* Correspondence: jbhur@korea.ac.kr
† This paper is an extended version of our paper published in the 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 7–10 January 2020.

Received: 30 April 2020; Accepted: 26 May 2020; Published: 29 May 2020

Abstract: Mail user agent (MUA) programs provide an integrated interface for email services. Many MUAs support email encryption functionality to ensure the confidentiality of emails. In practice, they encrypt the content of an email using email encryption standards such as OpenPGP or S/MIME, mostly implemented using GnuPG. Despite their widespread deployment, there has been insufficient research on their software structure and the security dependencies among the software components of MUA programs. In order to understand the security implications of the structures and analyze any possible vulnerabilities of MUA programs, we investigated a number of MUAs that support email encryption. As a result, we found severe vulnerabilities in a number of MUAs that allow cache side-channel attacks in virtualized desktop environments. Our analysis reveals that the root cause originates from the lack of verification and control over the third-party cryptographic libraries that they adopt. In order to demonstrate this, we implemented a cache side-channel attack on RSA in GnuPG and then conducted an evaluation of the vulnerability of 13 MUAs that support email encryption in Ubuntu 14.04, 16.04 and 18.04. Based on our experiment, we found that 10 of these MUA programs (representing approximately 77% of existing MUA programs) allow the installation of a vulnerable version of GnuPG, even when the latest version of GnuPG, which is secure against most cache side-channel attacks, is in use. In order to substantiate the importance of the vulnerability we discovered, we conducted a FLUSH+RELOAD attack on these MUA programs and demonstrated that the attack restored 92% of the bits of the 2048-bit RSA private key when the recipients read a single encrypted email.

Keywords: cache side-channel attack; encrypted email; mail user agent; GnuPG; desktop virtualization

1. Introduction

Because email is one of the most widely used communication services on the Internet [1], many Internet users are concerned about their communication privacy because sensitive email content can be revealed via bulk surveillance (e.g., illegal email surveillance by the NSA in the US) [2], social engineering attacks [3] or email service breaches [4–6]. Thus, a number of non-governmental organizations and social media platforms have recommended the adoption of security techniques to protect communications when sending sensitive information via email [7–12]. An example of this is email encryption, which was developed as a promising defense against email attacks. It secures the content of emails using end-to-end encryption in the application layer independently of transport layer

security (TLS). In order to preserve email privacy, many current mail user agent (MUA) programs support encrypted email functionality using email encryption standards such as OpenPGP [13] or S/MIME [14].

GnuPG (also known as GPG) [15] is a free implementation of the OpenPGP standard as defined by RFC4880 [13]. Since its introduction in 1997, the GnuPG crypto library has been widely used to implement OpenPGP in many MUA programs; however, though they are widely deployed in practice, there has been little research on the software structure, security dependencies among software components and security implications of MUA programs, especially in terms of cache side-channel attacks.

Since Gullasch et al. [16] first demonstrated the possibility of cache side-channel attacks, many variants have been proposed [17–20], and most of the secret-dependent cryptographic implementations that are vulnerable to cache side-channel attacks have been identified. For example, the RSA implementation of GnuPG 1.4.13 has a secret-dependent control flow. Thus, by monitoring the execution traces through a cache side-channel, an attacker is able to extract the RSA private key. The discovery of vulnerabilities in GnuPG led to the introduction of a security patch, thus it has been assumed that many of the applications utilizing GnuPG are secure against cache side-channel attacks because the problem was fundamentally resolved within the primitive crypto library; however, surprisingly, we found that many current MUAs are still vulnerable to cache side-channel attacks in cross-virtual machine (VM) environments sharing the same hardware, such as desktop virtualization environments.

In order to understand the root cause of this vulnerability, in this study, we conduct an in-depth analysis of the security mechanisms of existing MUA programs and determine how they take advantage of internal and external cryptographic libraries. We then propose a novel MUA attack scenario based on a cache side-channel attack and demonstrate that a number of MUAs are vulnerable to this attack. Our attack leverages the vulnerability of MUAs that allow the installation of an old version of the GnuPG library, which is vulnerable to cache side-channel attacks. An attacker can launch a cache side-channel attack when the victim reads an encrypted email in a VM environment. The attacker is consequently allowed to observe the cache activity of the victim and extract the RSA private key, which can be employed to restore the original content of the victim's email. We evaluate 13 existing MUAs that are available for use with Ubuntu 14.04 LTS, 16.04 LTS and 18.04 LTS. Based on our experiment, we find that 10 of these MUA programs are vulnerable to our attack scenario, even if the secure version of GnuPG is in use. In order to verify the severity of this vulnerability, we delivered a FLUSH+RELOAD attack on the MUA programs and demonstrated that an attack can restore 92% of the bits of the 2048-bit RSA private key when the recipients read a single encrypted email. Because encrypted email uses RSA encryption, the one-time extraction of a private key could compromise the privacy of all future emails.

Although cache side-channel attacks have been delivered against various TLS implementation vulnerabilities involving a variety of cryptographic algorithms and protocols, such as Lucky Thirteen [21] and Bleichenbacher [22], few studies have attempted to identify application vulnerabilities. Given that most of the current implementations of cryptographic libraries have been designed or patched to be secure against various cache side-channel attacks, it may be assumed that the applications using them would also be secure. As we demonstrate, however, an application's careless behavior or policies could incapacitate the underlying defense against these attacks. Thus, it is important for applications to avoid total dependency on the underlying defense provided by external security mechanisms; rather, they should carefully design and adopt their own independent security mechanisms.

Contribution. Our study makes the following contributions.

- We conduct an in-depth analysis of the security mechanisms implemented by existing MUA programs and identify their security flaws in terms of cache side-channel attacks in a virtualized desktop environment.

- We propose a novel MUA attack scenario based on a cache side-channel attack. It exploits an implementation flaw in MUA programs that allows a downgrade of the version of the cryptographic library used by the program. We find that many of these MUAs are vulnerable to this attack.
- To demonstrate the efficacy of the proposed cache-side channel attack in practice, we investigated the latest version of 13 existing MUA programs that work on Ubuntu 14.04 LTS, 16.04 LTS and 18.04 LTS. We find that 77% of these MUAs are vulnerable to our attack, which can extract 92% of the bits of the 2048-bit RSA private key from MUA users.
- We discuss mitigation strategies to secure MUA programs against cache side-channel attacks and explain why the security of the application layer should be independent of the underlying security of the other layers.

Though the presented attack scenario utilizes a FLUSH+RELOAD attack to assess the vulnerability, security misconfiguration and implementation flaws of the MUA programs, our scenario is not limited to this attack vector. In other words, more recent cache side-channel attacks such as PRIME+PROBE [18], EVICT+RELOAD [19] and FLUSH+FLUSH [20] can be used as attack vectors to demonstrate the efficacy of the proposed attack on the basis of the system environment involved.

Responsible disclosure. We have reported our findings to Evolution and disclosed the technical details to Evolution developers. Although we confirmed that the presented attack was possible due to the absence of control over GnuPG, which Evolution relies on, Evolution developers have claimed that it is a complex issue that combines both technical and human factors, which makes it difficult to easily resolve. Some of the technical feedback we obtained from Evolution has been integrated into the paper.

Organization. The rest of our paper is organized as follows. Preliminaries for MUA programs and cache side-channel attacks are given in Section 2. Our attack scenario, its demonstration on practical MUAs and analysis results are described in Section 3. Mitigation strategies are presented in Section 4. Related work is described in Section 5, followed by a conclusion and discussion in Section 6.

Extended part. Based on our previously published work [23], our paper includes further details to strengthen our attack scenario. Major extended parts are as follows. In Section 3, detailed results of analyzed practical MUAs and explanation for enforcing version downgrade of cryptographic library are included to consolidate practicality of our attack scenario. We discuss mitigation strategies considering the root cause that enables our attack scenario and real world user experiences in Section 4.

2. Preliminary

2.1. Mail User Agents

An MUA, also referred to as an email client, provides overall functionality for the access and management of a user's email. MUAs are preferred by many users because they can avoid the storage and functionality problems of web mail services and manage multiple email accounts in an integrated, ready-to-use email environment. Also, MUAs offer advantages in terms of reliability and security. For example, the emails stored in a personal system can be accessed whenever the user wants, even when the email server is shut down or network access fails. Many MUAs also provide email encryption functionality for secure email services using third-party crypto libraries such as OpenSSL or GnuPG. To access encrypted email, a corresponding secret key that is associated with the authorized user is required. In addition to basic email service functionalities, MUA programs that support email encryption also provide key management.

Thus, the protection of secret keys is the most important requirement for secure MUA email services; however, we have observed that many existing MUAs still allow the use of the CRT-RSA algorithm offered by GnuPG 1.4.13 (and earlier versions) to encrypt and decrypt emails by adopting the library as a plugin or add-on without a security check. Even in the presence of the patch for GnuPG (version 1.4.14 and later) introduced to prevent cache side-channel attacks, a flawed MUA

configuration can be exploited to avoid the security patch of the underlying crypto layer, as we demonstrate in Section 3.

2.2. Cache Side-Channel Attacks

Desktop virtualization. Desktop virtualization technology has been widely adopted by various organizations because of the lower management costs and higher security associated with separating the desktop environment from associated applications. Desktop virtualization technology centralizes hardware resources and assigns VMs to each user so that they can use the shared hardware (Figure 1). In a virtualized desktop system, all of the desktop components are virtualized, allowing for a highly flexible and secure desktop delivery model. In addition, because the components are saved in the data center, virtualization technology supports complete recovery if a user's device or hardware is lost. It offers numerous other advantages such as secure management, cost reduction and mobile computing [24]; however, in a virtualized environment, when individual users access separate desktop environments supported by their VM and use their personal devices to run independent applications, these applications are inherently executed using the shared hardware resources that manage the VMs. Thus, despite the advantages of virtualization technology, various cross-VM side-channel attacks [17,18,25] are possible by exploiting the intrinsic features of the shared hardware in the system, such as the L3 last level cache (LLC) in an Intel processor, unless the system employs appropriate prevention techniques.

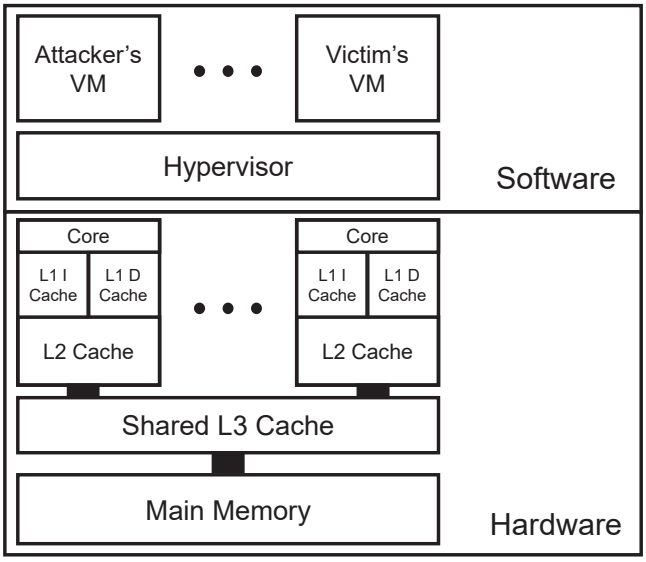


Figure 1. System model of the virtualized environment.

FLUSH+RELOAD attack. Memory deduplication is a useful strategy within a hypervisor system for efficient memory management. It allows processors to share a single copy of data instead of storing multiple copies. Even if the memory deduplication technique is sometimes disabled by current hypervisors by default for security reasons, it can be still activated by users explicitly because it is helpful to improve memory utilization, especially in a cross-VM environment. Yarom and Falkner [17] described how an attacker can exploit a pinhole in the feature for a successful side-channel attack, known as a FLUSH+RELOAD attack.

The FLUSH+RELOAD attack begins by flushing the observed data from the cache with a clflush instruction. The attacker then reloads the same data and measures the elapsed access time. In the period between flushing and reloading the data, if the victim loads the exact same data that the attacker is interested in (which will be reloaded later), a cache hit will occur during the attacker's reload phase,

which requires a shorter execution time compared to a cache miss. With a sufficient number of careful observations of the time difference, the adversary can figure out what data or function the victim accessed and in what order.

The FLUSH+RELOAD attack extends Gullasch et al.'s attack [16] in that (1) it uses the clflush instruction from Intel x86 architecture to flush the monitored memory lines from the LLC and probe the access patterns of specific memory lines and (2) it allows the spy and victim processes to be executed in parallel on different execution cores in the virtualized environment, which allows cross-VM attacks.

In order to demonstrate the efficacy of the side-channel attack, Yarom and Falkner showed how to extract the RSA private key of a victim running GnuPG 1.4.13 via the FLUSH+RELOAD attack. Specifically, because the access pattern of the memory lines for the square-and-multiply exponentiation algorithm in RSA depends on the value of the private key, the attacker can extract the RSA private key from the victim when he performs RSA decryption, even if the attacker and victim processes are running on separate VMs.

2.3. Cryptographic Implementation

The efficient implementation of cryptographic algorithms is of great importance in many cryptographic libraries such as OpenSSL and GnuPG. Therefore, many optimization techniques have been adopted to accelerate encryption or decryption operations in these libraries (for example, the square-and-multiply algorithm for RSA decryption). Unfortunately, these techniques can lead to unexpected vulnerabilities against side-channel attacks, such as FLUSH+RELOAD, as a side effect.

RSA implementation in GnuPG. GnuPG supports the RSA public-key algorithm for signature generation verification or data encryption/decryption in full support of OpenPGP. Because RSA decryption requires a significant number of exponentiation calculations, GnuPG adopts the optimization techniques CRT-RSA and square-and-multiply for efficient decryption in its implementation.

CRT-RSA. CRT-RSA is an optimization technique for rapid RSA decryption based on the Chinese Remainder Theorem (CRT). The general RSA algorithm proceeds as follows:

1. Randomly select two large prime numbers p and q.
2. Calculate $n = pq$.
3. Select e such that $1 < e < ((p-1)(q-1)-1)$ and $gcd(e, (p-1)(q-1)) = 1$, where $gcd(a, b)$ returns the greatest common divisor of a and b.
4. Calculate $d = e^{-1} \bmod (p-1)(q-1)$.
5. Set *public key* as (n, e), and *private key* as (d, n).

RSA encryption and decryption are performed by $Enc(m) : c \leftarrow m^e \bmod n$ and $Dec(c) : m \leftarrow c^d \bmod n$, respectively.

In CRT-RSA, the secret key d is split into two parts: $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$. The secret parameters would then be *private key* $= (d_p, d_q, p, q)$. The decryption function of CRT-RSA is described in Algorithm 1.

Algorithm 1: CRT-RSA Decryption

decryption(c, d_p, d_q, p, q)
$m_p \leftarrow c^{d_p} \bmod p$
$m_q \leftarrow c^{d_q} \bmod q$
$h \leftarrow (m_p - m_q)\,(q^{-1} \bmod p) \pmod p$
$m \leftarrow m_q + hq$
return m

Square-and-multiply algorithm. The square-and-multiply algorithm is used to reduce the number of exponent operations required for encryption and decryption. As shown in Algorithm 2, the square-and-multiply algorithm computes the exponentiation with a square–multiply operation if the bit of the binary representation of the exponent is 1 and computes it with a square operation

otherwise. For example, for a^{13}, the exponent 13 can be denoted as 1101_2. The operation sequence for the exponent would then be $(Square - Multiply, Square, Square - Multiply)$, each followed by a Modulo Reduce, instead of performing all exponentiation calculations, such as multiplying a 13 times. Because each iteration of the square and square–multiply operations exactly corresponds to each bit of the binary exponent, the execution flow contains all of the information about the exponents, which is the secret key in CRT-RSA.

Algorithm 2: Square-and-Multiply

```
// d_p is the binary representation of an exponent.
// n is the length of the d_p : (d_n, d_{n-1}, ..., d_0).
// (d_p)_i is the i^th bit of the d_p.
square and multiply(c, d_p, p)
m ← 1
for i = n − 1 down to 0 do
    m ← m^2 (mod p)
    if (d_p)_i == 1 then
        m ← mc (mod p)
    end
end
return m_p
```

Because decryption in CRT-RSA involves two exponents, acquiring the execution flow for the square-and-multiply algorithm in GnuPG allows the two elements d_p and d_q to be restored, which subsequently permits the full recovery of the victim's private key d. Yarom and Falkner [17] describes how the attacker is able to perform a FLUSH+RELOAD attack and collect the secret information of the private key when the victim performs RSA decryption in a virtualized desktop environment with GnuPG 1.4.13.

In later versions of GnuPG (1.4.14 or later), the square-and-multiply algorithm is replaced by the square-and-multiply-always algorithm to mitigate FLUSH+RELOAD attacks. The algorithm always performs a square–multiply operation for each bit and ignores the results of the multiply operation when the bit value is 0 [26]. Thus, the cache access patterns of each iteration contain access to memory lines for both the square-and-multiply operations at all times regardless of the bit value in the exponent, which hinders FLUSH+RELOAD attacks on GnuPG 1.4.14 and later versions.

3. Attacks on Mail User Agents

In this section, we analyze the vulnerabilities of MUA programs in the real world and present our attack scenario against the MUAs to recover the RSA secret key of a victim. In our attack scenario, we assume that the attacker and victim processes are deployed in the same virtualized desktop environment, in which the VMs adopt page-sharing.

The process of our attack consists of the following three steps. First, the attacker performs a FLUSH+RELOAD attack when a victim reads an encrypted email via the MUA. In this step, the attacker observes the victim's execution of the decryption algorithm provided by GnuPG and acquires bit information for the RSA private key of the victim. Second, the adversary restores the private key based on the observed bit information in the first step. Finally, the adversary decrypts the victim's encrypted email in the MUA with the restored private key.

Figure 2 provides an overview of our attack scenario. The red arrows in the figure depict the overall process of our attack, which seeks to restore the RSA private key of the victim. The blue arrows show the process of acquiring an encrypted email delivered to the victim and decrypting the encrypted content using the restored private key.

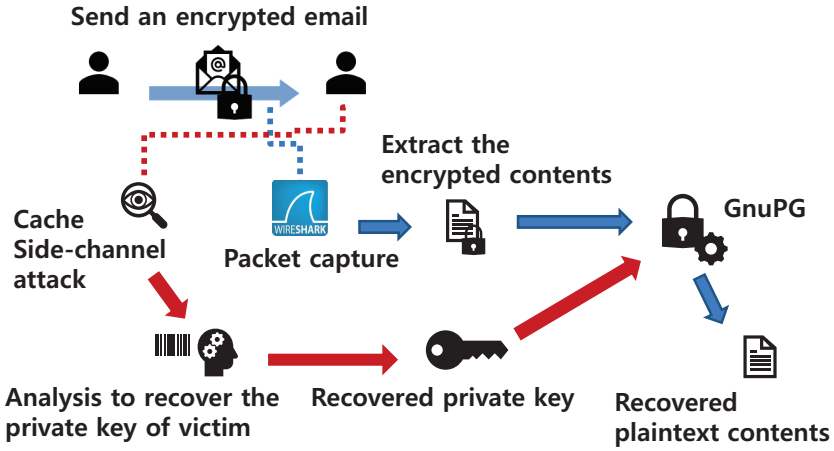


Figure 2. Overview of our attack scenario.

3.1. Analyzing Mail User Agents

We first investigated 37 MUA programs supporting email encryption functionality and installed 13 of these on each VM, all of which are available in Ubuntu 14.04, 16.04 and 18.04. We then tested whether they allowed the installation of GnuPG 1.4.13, which is vulnerable to FLUSH+RELOAD attacks. As shown in Table 1, 10 of these MUAs allowed installation without checking the version of the crypto library, making our attack feasible. However, the other three MUAs (Thunderbird, Seamonkey Mail and Sylpheed) blocked the installation by checking the version of the third-party crypto library. Based on our experiment, these three MUAs only allowed the installation of GnuPG 2.0.14 or later.

Table 1. Vulnerability Analysis of mail user agents (MUAs).

	Ubuntu 14.04		**Ubuntu 16.04**		**Ubuntu 18.04**	
MUA	**Version**	**GnuPG 1.4.13**	**Version**	**GnuPG 1.4.13**	**Version**	**GnuPG 1.4.13**
Alpine	2.10	Allowed	2.20	Allowed	2.21	Allowed
Balsa	-	-	2.5.6	Allowed	2.5.3	Allowed
Claws Mail	3.9.3	Allowed	3.13.2	Allowed	3.16.0	Allowed
Cone	0.89	Allowed	1.0	Allowed	-	-
Evolution	3.10.4	Allowed	3.18.5.2	Allowed	3.28.5	Allowed
GNUmail	-	-	1.2.2	Allowed	-	-
i.Scribe	-	-	2.2	Allowed	2.3	Allowed
KMail	4.13.3	Allowed	5.1.3	Allowed	5.7.3	Allowed
Thunderbird	52.9.1	Prevented	60.2.1	Prevented	60.2.1	Prevented
Seamonkey	2.49.4	Prevented	2.49.4	Prevented	2.49.4	Prevented
Sylpheed	3.6	Prevented	3.5	Prevented	3.5.1	Prevented
Trojitá	0.7	Allowed	-	-	-	-
Zimbra	7.3.1	Allowed	-	-	-	-

Thunderbird uses the Enigmail plugin for GnuPG. We installed Thunderbird 52.9.1 on Ubuntu 14.04 and Thunderbird 60.2.1 on Ubuntu 16.04 and 18.04. When we executed Thunderbird in all Ubuntu versions, we observed a warning message from the MUA program about the potential installation of an old version of GnuPG, which disallowed the sending or retrieving of encrypted email. Thus, the cache side-channel attack could not obtain any traces with either version of Thunderbird. **Seamonkey Mail**, and **Sylpeed** also prevented the use of all encrypted email services using a vulnerable version of GnuPG in Ubuntu 14.04, 16.04 and 18.04.

Alpine, **Claws Mail**, **Evolution** and **Kmail** allowed the installation of GnuPG 1.4.13 in all three versions of Ubuntu in full support of encrypted email services using GnuPG. During the installation of

the vulnerable libraries on the MUA and the sending/retrieving of emails via the MUA, no warning messages were generated.

Cone in Ubuntu 14.04 and 16.04 and **Balsa** and **i.Scribe** in Ubuntu 16.04 and 18.04 allowed the installation of GnuPG 1.4.13 and provided all of the functionality of encrypted email without warning the user about the old version of the crypto library.

Trojita and **Zimbra** in Ubuntu 14.04 and **GNUmail** in Ubuntu 16.04 also allowed the installation of the vulnerable crypto library and supported full email encryption services without any warning messages or safeguard mechanisms.

3.2. Configuring the Attack

We designed our attack based on the FLUSH+RELOAD technique, assuming that the attacker and the victim each have a VM in a shared data center. We used a server equipped with an Intel Xeon E5-2620v4 processor and 256 GB of RAM, running the QEMU-KVM hypervisor (1:2.5+dfsg-5ubuntu10.5). In our experiment, two guest VMs with Ubuntu OS were instantiated on top of the hypervisor. On each VM, we installed the 1.4.13 version of GnuPG and the latest available version of each MUA.

Enforcing version downgrade of cryptographic library. It may be assumed that forcing the victim to install an older version of GnuPG is impractical in the real world; however, there are several threat models that make it practically feasible. For example, adversaries may be able to persuade the victim to download a vulnerable version of the crypto library from a fake GnuPG website by delivering a DNS spoofing attack or a watering hole attack [27]. As another example, an algorithm substitution attack (also known as a subversion attack) [28] is also an effective attack vector to replace an honest implementation of a cryptographic tool with a subverted one. It can force a leak of private information by having unaware users install a subverted cryptographic library, with the generated output indistinguishable from the honest output. In addition, there is a diverse array of social engineering phishing attacks that can lure users into downloading fake software. Any of these methods could lead to a version downgrade of the cryptographic library in the real world.

Implementing cache side-channel attack. Because the cache side-channel attack is dependent on the hardware environment, we adjusted several features of our experiment system, such as the addresses to probe and the threshold and slot size for the detection of cache hits/misses, in order to configure our attack scenario for the MUA programs.

We used a gdb debugger to determine the addresses of the square, multiply and reduce operations of CRT-RSA in the GnuPG execution file, which will be probed by the attacker to obtain the execution patterns for decryption when the victim reads an encrypted email. We then measured the access time of approximately 80,000 square-and-multiply operations of CRT-RSA to set the threshold. We also measured the elapsed time of the overall probing process to set the slot size. As a result, we set 120 cycles as the threshold and 5000 cycles as the slot size based on the success ratio of the measurements.

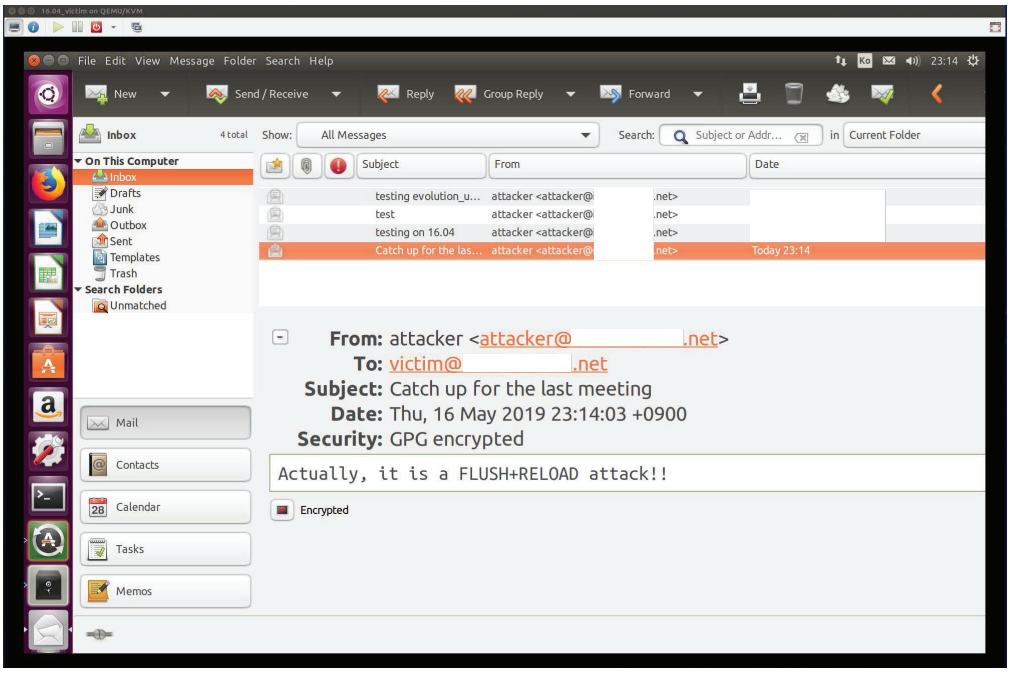
3.3. Attacking the Mail User Agent

Based on the analysis results in Table 1, we delivered our attack on vulnerable MUAs. The attacker runs the spy process for a FLUSH+RELOAD attack then sends an encrypted email to the victim. At this time, the adversary's MUA encrypts the email content with the public key of the victim. When the victim retrieves the encrypted email, the MUA decrypts the content by executing the RSA algorithm in GnuPG, the vulnerability of which is exploited by the adversary's spy process.

We implemented our attack scenario on the Evolution MUA, which is one of the vulnerable MUAs that we previously identified. Figure 3a shows the attacker's view when launching a FLUSH+RELOAD attack, and Figure 3b shows the victim's view during email retrieval. In the attacker's view, S and M represent the square-and-multiply operations, respectively, and r is the reduce operation preparing the next input, which is the next execution of the square-and-multiply operation. This experiment

demonstrates that the attacker is able to obtain the execution flow of the square-and-multiply operation of the RSA algorithm during email retrieval, which is used to restore the victim's RSA secret key.

(**a**)



(**b**)

Figure 3. Attack on Evolution MUA: (**a**) Attacker's view. (**b**) Victim's view.

3.4. Recovering the RSA Key

To restore the full RSA decryption key, we need to convert the execution flow of the square-and-multiply operation obtained in Section 5.3 to the corresponding values of the RSA key exponents. Because CRT-RSA decryption in GnuPG 1.4.13 (or previous versions) involves two exponents, tracing the execution flow of the square-and-multiply operation results in the two elements

d_p and d_q being restored. d_p and d_q are sufficient to lead full key recovery [29]. In order to calculate the victim's secret key from these two values, we implemented a converter which converts the sequences 'square-reduce-square' and 'square-and-square' to 0 and 'square-reduce-multiply' and 'square-multiply' to 1. In the square-and-multiply algorithm, even if the conditional reduce operation is adopted for efficient exponentiation, it does not actually affect the RSA key exponent. This motivated us to remove the reduce operations in the trace process, which could remove approximately a third of all probing cases. Because, in practice, each probe operation may inevitably be affected by noise in the system, reducing the number of probes would be very helpful for improving the accuracy of key restoration. By taking advantage of this technique in our converter, the attacker could recover 92% of the bits of the victim's 2048-bit RSA secret key for the best case in the proposed attack scenario. To determine the rest of the bits, the attacker can simply conduct the attack multiple times and combine the results, or leverage several existing strategies such as reducing search space for brute force attack, and applying RSA exponent recovery algorithm from partial information [17]. Because the objective of our experiment was to demonstrate the vulnerability of the MUA programs that we found were capable of being exploited, determining the rest of the decryption key or improving the accuracy of the restoration is outside of the scope of this paper.

3.5. Decrypting Email

We observed that there are two different ways to acquire an encrypted email from the victim. Because the attacker and the victim share the same hardware, the attacker can easily capture all of the network traffic of the victim using a packet-capturing tool. In our experiment, we had an outsider send an encrypted email to the victim, which the attacker then obtained the packets using the Wire Shark packet-capture program. As an alternative to capturing network traffic, another possible way to obtain encrypted email is to obtain it as an attached file from the web email service. When a user sends an encrypted email using an MUA program, the receiver retrieves it using other mail programs, such as web mail services; we found that the encrypted email is supposed to be delivered as a separate file rather than as email content. Thus, if the adversary compromises the victim's account by some means, it would be possible to obtain the attached encrypted email by simply downloading it. If the adversary succeeds in acquiring an encrypted email from the victim, it would be very straightforward to decrypt it using the restored decryption key. Even if the latter case requires additional effort to compromise the victim's account, the former case does not require such effort, and easily can be achieved by simple wire-tapping.

4. Mitigation

It is widely believed that, if vulnerabilities are fundamentally resolved in the crypto library, applications would be secure as long as they rely on the library for their security; however, as many previous studies have demonstrated [21,30–32], this belief may be unfounded in practice because applications themselves may have unexpected vulnerabilities or implementation flaws that allow attackers to revive old vulnerabilities of the libraries even if they have been patched.

Verification of underlying library. One important root cause that allows our attack to occur is that several MUAs completely depend on the underlying crypto libraries for their email security; they do not implement any security mechanisms of their own (such as verification of the crypto library via a simple version check). Hence, a simple and effective mitigation of this attack would be for the MUAs to determine if they have any implementation or configuration flaws and to adopt security mechanisms (at least, for example, a more rigorous security verification mechanism for the crypto library before installation) if such flaws are discovered.

Application side security mechanism and user experience. In terms of usable security, despite the efforts of MUA developers, users still struggle to understand how to use encrypted email services securely [33–35]. As previous studies have reported, it is difficult to expect users to follow the recent security patches for the underlying crypto libraries in the real world. Therefore, it is important to note that programs utilizing outer layer libraries for their security must somehow analyze whether they

are secure at the time of installation. If not, the programs should prevent installation or utilization on behalf of the users.

5. Related Work

5.1. Cross-VM Cache Side-Channel Attacks

In 2009, Ristenpart et al. [36] showed that the use of cloud virtualization could introduce vulnerability to cross-VM side-channel attacks, which allow attackers to extract secret information from a target VM on the same machine. They demonstrated the efficacy of the attack scenario in practical cloud systems, such as Amazon EC2. In 2011, Suzaki et al. [37] demonstrated that memory deduplication of kernel same-page merging (KSM) on a KVM hypervisor is vulnerable to memory disclosure attacks and explored how this is exploited to identify and restore the data of another user as a form of cross-VM side-channel attack. Gullasch et al. [16] presented another cache side-channel attack on AES, which enables an attacker to exploit the cache behaviors of processors on shared memory pages in a single-core processor and to recover the full secret key of AES-128. They suggested a technique that exploited the Intel x86 clflush instruction to flush the monitoring memory location and then to determine whether the data was cached to that location to extract access patterns for the victim process. In 2014, Yarom and Falkner [17] proposed the FLUSH+RELOAD cross-VM cache side-channel attack based on the technique introduced by Gullash et al. Unlike previous cache side-channel attacks, however, they demonstrated that the LLC in Intel x86 architecture can be used as a covert channel to extract secret information from other victim programs when memory pages are shared between processes (This inevitably happens when memory deduplication is adopted in the hypervisor.).

5.2. Attacks on Encrypted Email

In 2001, Davis [38] analyzed the security of email encryption and signature standards such as S/MIME, PGP, PEM, MOSS and PKCS#7, and proposed a surreptitious forwarding attack exploiting naive sign and encrypt flaws in the standards. Poddebniak et al. [39] described a plaintext injection attack on encrypted email that exploits the malleability of CBC/CFB in OpenGPG and S/MIME in 2018. The attack reveals the plaintext of encrypted email by generating an exfiltration channel when a recipient decrypts an encrypted email to read. They analyzed practical MUA programs, webmail and web apps based on OpenPGP and S/MIME encryption and identified what exfiltration channels can be established in many existing email client programs. Because this vulnerability was found in standard conforming implementations, they advised that the standards be updated to mitigate the attack.

5.3. Attacks on Cryptographic Algorithms and Protocols

In 2015, Irazoqui et al. [21] reanimated the patched vulnerability of the TLS and DTLS protocol libraries [30] by exploiting their implementation flaws, known as the Lucky Thirteen attack. In their research, they used a FLUSH+RELOAD attack by exploiting memory deduplication in a hypervisor to reestablish the patched vulnerability in a cross-VM environment. Recently, in 2018, Ronen et al. [40] revived the Lucky Thirteen vulnerability and explored how the current implementation patch for TLS libraries is still vulnerable to another type of cache side-channel attack known as Prime+Probe. Similarly, Böck et al. [32] demonstrated that many existing email programs in the wild are vulnerable to a Bleichenbacher attack [31], which exploits the side-channel vulnerability of PKCS#1 v.1.5 in TLS libraries, even if a countermeasure to the Bleichenbacher attack is already known in practice.

Although many side-channel attacks have been delivered to exploit implementation vulnerabilities in a diverse range of cryptographic algorithms and protocols, few studies have attempted to identify MUA vulnerabilities against cache side-channel attacks in practice.

6. Conclusions and Discussion

In this study, we analyzed the security mechanisms of existing MUAs and identified a security flaw that allows a version downgrade of the cryptographic library due to a lack of verification and control. We then proposed a novel cache side-channel attack scenario exploiting the vulnerability of MUAs and demonstrated that a number of existing MUAs are still vulnerable to the attack. In addition to the MUA programs we investigated in the present study, other applications can be exposed to the same threat if they have similar implementation flaws. As we demonstrated in this study, implementing security mechanisms in the application layer that are independent of the underlying security in the other layers is of utmost importance for MUA security.

Author Contributions: Investigation, H.K.; methodology, H.K. and Y.S.; software, H.K. and H.Y.; supervision, J.H.; writing—original draft, H.K.; Writing—review and editing, Y. S. and J.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the research fund of Signal Intelligence Research Center supervised by the Defense Acquisition Program Administration and Agency for Defense Development of Korea.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Radicati Group, Inc. *Email Statistics Report, 2018–2022*; Technical Report; Palo Alto, CA, USA, 2018.
2. Landau, S. Making sense from Snowden: What's significant in the NSA surveillance revelations. *IEEE Secur. Priv.* **2013**, *11*, 54–63.
3. Egelman, S.; Cranor, L.F.; Hong, J. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In Proceedings of the 26th ACM Conference on Human Factors in Computing Systems (CHI), Florence, Italy, 5–10 April 2008; pp. 1065–1074.
4. VP Contender Sarah Palin Hacked. Wikileaks. 2008. Available online: https://wikileaks.org/wiki/VP_contender_Sarah_Palin_hacked (accessed on 27 May 2020).
5. Sony Email Archive. Wikileaks. 2015. Available online: https://wikileaks.org/sony/ (accessed on 27 May 2020).
6. Hillary Clinton Email Archive. Wikileaks. 2016. Available online: https://wikileaks.org/clinton-emails/ (accessed on 27 May 2020).
7. VerschlüSselte Kommunikation via PGP Oder S/MIME. Amnesty International. Available online: https://www.amnesty.de/keepitsecret (accessed on 27 May 2020).
8. *Safety Guide for Journalists—A Handbook for Reporters in High-Risk Environments*; United Nations Educational, Scientific and Cultural Organization and Reporters Without Borders: Paris, France, 2017.
9. Got a Confidential News Tip? The New York Times. Available online: https://www.nytimes.com/tips?module=inline (accessed on 27 May 2020).
10. Confidential Tips—Maximize Your Data Security. The Washington Post. Available online: https://www.washingtonpost.com/anonymous-news-tips/?utm_term=.feb659746de3 (accessed on 27 May 2020).
11. How to Contact the Guardian Securely. The Guardian. Available online: https://www.theguardian.com/help/2016/sep/19/how-to-contact-the-guardian-securely (accessed on 27 May 2020).
12. How to: Use PGP for Windows. Electronic Frontier Foundation. Available online: https://ssd.eff.org/en/module/how-use-pgp-windows (accessed on 27 May 2020).
13. *OpenPGP Message Format*; RFC4880; ITFE: Fremont, CA, USA, 2007.
14. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC5751. Available online: https://tools.ietf.org/html/rfc5751 (accessed on 27 May 2020).
15. The GNU Privacy Guard (GnuPG). GNU Project. https://www.gnupg.org (accessed on 27 May 2020).
16. Gullasch, D.; Bangerter, E.; Krenn, S. Cache games–Bringing access-based cache attacks on AES to practice. In Proceedings of the 32nd IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 22–25 May 2011; pp. 490–505.
17. Yarom, Y.; Falkner, K. FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security), San Diego, CA, USA, 20–22 August 2014; pp. 719–732.

18. Liu, F.; Yarom, Y.; Ge, Q.; Heiser, G.; Lee, R.B. Last-Level Cache Side-Channel Attacks are Practical. In Proceedings of the IEEE Symposium on Security & Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 605–622.
19. Daniel Gruss, R.S.; Mangard, S. Cache template attacks: Automating attacks on inclusive last-level caches. In Proceedings of the 24th USENIX Security Symposium, Washington, DC, USA, 12–14 August 2015; pp. 897–912.
20. Gruss, D.; Maurice, C.; Wagner, K.; Mangard, S. Flush+Flush: A fast and stealthy cache attack. In Proceedings of the 13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, San Sebastián, Spain, 7–8 July 2016; pp. 279–299.
21. Irazoqui, G.; Inci, M.S.; Eisenbarth, T.; Sunar, B. Lucky 13 strikes back. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (AsiaCCS), Singapore, 14–17 April 2015; pp. 85–96.
22. Ronen, E.; Gillham, R.; Genkin, D.; Shamir, A.; Wong, D.; Yarom, Y. The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations. In Proceedings of the 40th IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–22 May 2019.
23. Kim, H.; Yoon, H.; Shin, Y.; Hur, J. Cache Side-Channel Attack on Mail User Agent. In Proceedings of the 2020 International Conference on Information Networking (ICOIN), Barcelona, Spain, 7–10 January 2020; pp. 236–238.
24. Yan, L. Development and application of desktop virtualization technology. In Proceedings of the 2011 IEEE International Conference on Communication Software and Networks (ICCSN), Xi'an, China, 27–29 May 2011; pp. 326–329.
25. Zhang, Y.; Juels, A.; Reiter, M.K.; Ristenpart, T. cross-VM side channels and their use to extract private keys. In Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS), Raleigh, NC, USA, 16–18 October 2012; pp. 305–316.
26. Coron, J.S. Resistance against differential power analysis for elliptic curve cryptosystems. In Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Worcester, MA, USA, 12–13 August 1999; pp. 292–302.
27. Alrwais, S.; Yuan, K.; Alowaisheq, E.; Liao, X.; Oprea, A.; Wang, X.; Li, Z. Catching Predators at Watering Holes: Finding and Understanding Strategically Compromised Websites. In Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC), Los Angeles, CA, USA, 5–9 December 2016; pp. 153–166.
28. Sebastian Berndt, M.L. Algorithm Substitution Attacks from a Steganographic Perspective. In Proceedings of the ACM Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1649–1660.
29. Campagna, M.J.; Sethi, A. Key Recovery Method for CRT Implementation of RSA. *IACR Cryptol. ePrint Arch.* **2004**, *2004*, 147.
30. Al Fardan, N.J.; Paterson, K.G. Lucky thirteen: Breaking the TLS and DTLS record protocols. In Proceedings of the 34th IEEE Symposium on Security and Privacy (SP), Berkeley, CA, USA, 19–22 May 2013; pp. 526–540.
31. Bleichenbacher, D. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In Proceedings of the 18th Annual International Cryptology Conference (CRYPTO), Santa Barbara, CA, USA, 23–27 August 1998; pp. 1–12.
32. Böck, H.; Somorovsky, J.; Young, C. Return Of Bleichenbacher's Oracle Threat (ROBOT). In Proceedings of the 27th USENIX Security Symposium (USENIX Security), Baltimore, MD, USA, 15–17 August 2018; pp. 817–849.
33. Whitten, A.; Tygar, J.D. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In Proceedings of the 8th USENIX Security Symposium (USENIX Security), Washington, DC, USA, 23–36 August 1999; Volume 348.
34. Sheng, S.; Broderick, L.; Koranda, C.A.; Hyland, J.J. Why johnny still can't encrypt: Evaluating the usability of email encryption software. In Proceedings of the Second Symposium On Usable Privacy and Security (SOUPS), Pittsburgh, PA, USA, 12–14 July 2006; pp. 3–4.
35. Ruoti, S.; Andersen, J.; Zappala, D.; Seamons, K. Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *arXiv* **2015**, arXiv:1510.08555.

36. Ristenpart, T.; Tromer, E.; Shacham, H.; Savage, S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), Chicago IL, USA, 9–13 November 2009; pp. 199–212.
37. Suzaki, K.; Iijima, K.; Yagi, T.; Artho, C. Memory deduplication as a threat to the guest OS. In Proceedings of the 4th European Workshop on System Security (EuroSec), Salzburg, Austria, 10 April 2011; p. 1.
38. Davis, D. Defective Sign & Encrypt in S/MIME, PKCS# 7, MOSS, PEM, PGP, and XML. In Proceedings of the 2001 USENIX Annual Technical Conference, General Track, Boston, MA, USA, 25–30 June 2001; pp. 65–78.
39. Poddebniak, D.; Dresen, C.; Müller, J.; Ising, F.; Schinzel, S.; Friedberger, S.; Somorovsky, J.; Schwenk, J. Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels. In Proceedings of the 27th USENIX Security Symposium (USENIX Security), Baltimore, MD, USA, 15–17 August 2018; pp. 549–566.
40. Ronen, E.; Paterson, K.G.; Shamir, A. Pseudo constant time implementations of TLS are only pseudo secure. In Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS), Toronto, ON, Canada, 15–19 October 2018; pp. 1397–1414.